(triangle size "solid"/"outline" color)
(beside im1 im2 ...)
(above im1 im2 ...)

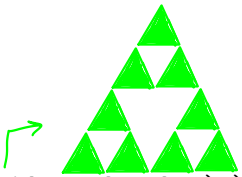
Complete the missing expressions below

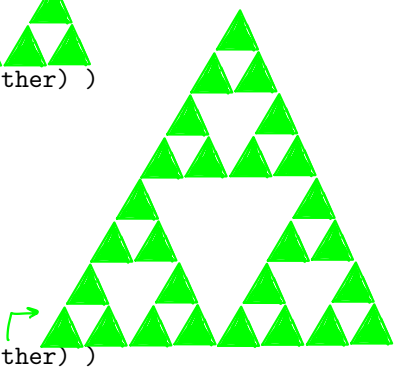
```
(require picturing-programs)


; sierpinski triangle of depth 0
(define sierp_0 (an expression for a solid green triangle of size 10) )
```

```
; sierpinski triangle of depth 1
(define sierp_1 (an expression for sierp_0 above two sierp_0s beside each other) )
```

```
; sierpinski triangle of depth 2
(define sierp_2 (an expression for sierp_1 above two sierp_1s beside each other) )
```





```
; sierpinski triangle of depth 3
(define sierp_3 (an expression for sierp_2 above two sierp_2s beside each other) )


; sierpinski triangle of depth 4
(define sierp_4 (an expression for sierp_3 above two sierp_3s beside each other) )
```

define function
invoke, use function

Complete the missing parts of the function sierp below.

```
(require picturing-programs)

; sierp : number -> image
; Sierpinski's triangle of depth d
(define (sierp d)
  (cond
    [(zero? d)
      (here you need an expression for a solid green triangle of size 10) ]
```

(triangle 10 "solid" "green")

```
    [(equal? d 1)
       (an expression for (sierp 0) above two (sierp 0)s beside each other) ]

    [(equal? d 2)
       (an expression for (sierp 1) above two (sierp 1)s beside each other) ]

    [(equal? d 3)
       (an expression for (sierp 2) above two (sierp 2)s beside each other) ]

    [(equal? d 4)
       (an expression for (sierp 3) above two (sierp 3)s beside each other) ]

    ))
```

The definition of `sierp` was a bit repetitive, and only went as far as allowing `(sierp 4)`. Use the same ideas, but do some arithmetic with the placeholder `d` to define `sierpinski` below:

```
(require picturing-programs)


(define (sierpinski d)
  (cond
    [(zero? d) (triangle 10 "solid" "green")]
    [else
      (an expression to put a sierpinski of one smaller than d above
       two sierpinskis of one smaller than d)  ]







    ))
```